REMARKS/ARGUMENTS

Claims 1-26 were pending in the present application when last examined. Claims 27-29 have been added. No claims have been amended and no new matter has been added. Support for the new claims is given below. Therefore, upon entry of this amendment, claims 1-29 will be pending.

Claim Rejections under 35 USC § 103

Claims 1-26 have been rejected under 35 U.S.C. 103(a) as being unpatentable by Lin, U.S. Patent Application Publication No. US 20050071345 (hereinafter "Lin"), in view of Govindarajan et al., U.S. Patent Application Publication no. US 20020174128 (hereinafter "Govind"). Applicants respectfully traverse and request withdrawal of this rejection for at least the following reasons below.

Lin in view of Govind fails to disclose storing the data values of first and second fields to a single column in the data structure

Lin in view of Govind fails to teach or suggest "storing the data values of first and second fields to a *single column* in the data structure, wherein the single column includes data values that may include *different data types* for different tenants" as is claimed in claim 1. (emphasis added) Independent claims 5, 9, and 20-23 all recite substantially similar claim language directed to embodiments of data structures for multiple tenants, and thus these claims are also not rendered obvious by Lin in view of Govind for at least the same reasons as claim 1. All of the remaining claims are dependent from one of the above independent claims, and thus are also allowable for at least the same reasons.

The Examiner states in the Office Action that Lin does not disclose "storing the data values of first and second fields to a *single* column in the data structure," and cites to Govind at ¶84 address this claim element. However, Govind also fails to teach this claim element. While Govind teaches how a user can create multiple custom fields in a *single data*

structure, Govind does not teach or suggest, and in fact teaches away from, multiple custom fields that exist in a *single column*.

Govind at ¶84 discloses that a user may register a new data type with a database, including the name, attributes, and pickling/unpickling routines for the new data type. However, this passage of Govind does not disclose that any pair of new data types (or existing data types) can be combined in a single column of a data structure.

Govind does discuss, at places such as its Background section (¶¶32-45), the use of "Raw" data types to help define custom database objects. However, a database table, and any associated column within that table that uses a custom data type, must still be defined as a single data type. While giving examples of how custom data types can be stored, the Background section of Govind states "[t]he following statements create a table with *a RAW column* that can be used, for example, *for storing data from a TYPE1 data items*." [Emphasis added] (¶33) Thus, Govind discloses the creation of a single column for use with a single custom data type.

Govind goes on to state that the use of "Raw" columns is not ideal because "[c]onsequently, the database system would not be able to detect situations in which one user *erroneously* stores data from one kind of user-defined data type in a RAW column that is supposed to hold data from a different kind of user-defined data type." [Emphasis Added] (¶49) Thus, Govind clearly teaches away from combining two different data types in a single data column because this could result in the corruption of data within a "Raw" field.

For example, suppose that a user defines a first data type consisting of a number, a text field, and pickling routines. This data type can be called "Type1." Further suppose that a user defines a second data type consisting of a number, a text field, a date, and pickling routines. This data type can be called "Type2." Based on the above, Govind thus considers the example shown in Table 1, shown below, as an error state:

ID	Data
1	{Type 1}
2	{Type 1}
3	{Type 2}

Table 1

However, the present claims are directed toward embodiments that cover the exact scenario in Table 1. Also see Fig. 7 in the present disclosure for a more detailed example of the scenario described in Table 1.

Reading the disclosure of Govind, one skilled in the art would only be able to recognize that tables such as the one below could be created using the teaching of Govind:

ID	Field 1	Field 2
1	{Type 1}	{Type 2}
2	{Type 1}	{Type 2}
3	{Type 1}	{Type 2}

Table 2

Since Lin and Govind do not disclose or teach, "storing the data values of first and second fields to a *single* column in the data structure," the Applicants respectfully submit that claims 1-26 are patentable over Lin and Govind for least the foregoing reasons. Withdrawal of the present rejections of claims 1-26 is respectfully requested.

Lin in view of Govind fails to disclose indexing of custom data fields

In addition to the claim elements contained in the independent claims of the Applicant's application, Lin in view of Govind fails to teach or suggest "copying to a first one of the *index columns* the data values stored in the single data column for the first field *in response to a request from the first tenant* to *index data* in the first data field" as is claimed in claim 3. (emphasis added) Claims 4, 25, and 26 all trace their dependence back to claim 3, and thus also not anticipated by Lin for at least the same reasons as claim 3. Similarly, claims 6-7 and 13-15 either directly recite or are dependent from claims that recite the indexing of custom fields.

A user-created database index, as the term is commonly used in the art, is a structure within a database that improves the performance of the database for certain operations.[1] This is typically accomplished by the database by making and using additional copies of frequently accessed data fields or columns. The present claims allow for a tenant to designate custom fields that should be indexed in the tenant's virtual database in much the same way a database administrator might index fields in a standard database. This feature allows for a tenant to improve the performance of his virtual private database using the tenant's own custom data fields.

The Examiner cites ¶40 of Lin in the Office Action as disclosing the specific features of claim 3. However, this paragraph only discusses how data is copied between tables when an application sitting on top of the data repository is upgraded. In fact, Lin states in related ¶35 that "techniques are provided for upgrading an application without losing previously-made user-customizations to the data repository that is used by the application...the data for the custom attributes stored in these tables is not overwritten or lost during an upgrade." ¶40 further describes aspects of this functionality.

It is clear that ¶40 has *nothing* to do with the ability of the system to allow a tenant to index custom data fields to improve system performance. The problem addressed by ¶40 of Lin concerns the ability of Lin to preserve existing custom fields when the database structure itself is being upgraded. The creation of an index allows for improved performance of a database while maintaining the same basic underlying structure. There generally are not any concerns about losing data when creating an index in a database, and so ¶40 does nothing to teach or suggest the creation of database indexes on custom fields.

Furthermore, a database key is not the same thing as a user-defined index. A key, as is commonly used in the art, refers to a data value that can uniquely identify a given record. As one example, an "ID" can be given to each record in a table, and if the ID is unique for each record, then the ID is a key. An index is a user-defined optimization of a database, and an indexed data field has no requirement that each value of the data field uniquely identify a record.

[1] *See, e.g.*, http://en.wikipedia.org/wiki/Index_(database). ("A database index is a data structure that improves the speed of operations on a database table.")

Govind also does not teach anything relating to the indexing of custom data fields, and the Examiner does not cite Govind as teaching this particular element of the claims. In fact, Govind may even teach away from the ability to index custom data fields because Govind teaches the ability to store data in "raw" data formats not understood natively by the database (*see, e.g.*, Abstract). As such, Govind cannot address the deficiencies in the disclosure of Lin.

Lin in view of Govind simply fails to teach or suggest any functionality related to a user's ability to index data fields, and thus cannot be said to render obvious claims 3-4, 6-7, 13-15, and 25-26. It is therefore respectfully submitted that claims 3-4, 6-7, 13-15, and 25-26 are patentable over Lin in view of Govind for at least the foregoing reasons. Withdrawal of the rejection is respectfully requested.

Lin in view of Govind fails to disclose a multi-tenant data structure

Lin in view of Govind fails to teach or suggest "a *multi-tenant* data structure having a plurality of data columns and one or more index columns" as is claimed in claim 1. (emphasis added) Lin in view of Govind also fails to teach or suggest a single column that "includes data values that may include different data types *for different tenants*" as is claimed in claim 1. (emphasis added) Independent claims 5, 9, and 20-23 all recite substantially similar claim language directed to embodiments of data structures for multiple tenants, and thus these claims are also not rendered obvious by Lin in view of Govind for at least the same reasons as claim 1. All of the remaining claims are dependent from one of the above independent claims, and thus are also allowable for at least the same reasons.

The Examiner cites ¶28 of Lin as teaching a multi-tenant data structure; however, this section of Lin only teaches a custom attribute table. In the Advisory Action mailed on December 31, 2007 Advisory Action, the Examiner further stated that:

> Lin may not explicitly use the term multi-tenant database. But Lin nonetheless discloses a mult-tenant datastructure by its functionality. The custom attributes disclosed in Lin allow for the exact same functionality as a multi-tenant database structure. The custom attributes allow for multiple customer organizations to share database resources which is the essence of a multi-tenant database structure.

The Applicants respectfully disagree that custom attributes, either as disclosed by Lin or as otherwise disclosed in *any* reference, provide the "exact same functionality as a multi-tenant database structure" as asserted by the Examiner. A multi-tenant data structure can exist without the ability to create custom attributes, and a data structure that can create custom attributes can exist without the ability to service multiple tenants. The reason for this is that a multi-tenant data structure concerns the *permissions* granted to various tenants while custom attributes concern the ability to *customize data* in the data structure.

One aspect of a multi-tenant data structure, as is described in the Applicant's disclosure, is that "tenant data is preferably arranged so that data of one tenant is kept separate from that of other tenants so that one tenant does not have access to another's data, unless such data is expressly shared." (¶0023) This feature makes intuitive sense because "a multi-tenant architecture is used wherein customer organizations (i.e., tenants) share database resources in one logical database. The database tables themselves are typically shared; each entity in the data model typically contains an organization_id column that distinguishes rows for each tenant. All queries and data manipulation in the context of a tenant filter on this (indexed) organization_id column to ensure proper security and the appearance of virtual private databases." (¶0002) When multiple tenants, who will often be unknown to each other, are all using the same logical database to create and maintain their own virtual private databases, the system needs to implement methods to ensure that data is not improperly shared between tenants. This problem becomes more difficult when data from different tenants is stored not only in common tables, but often in common columns within those tables. If a multi-tenant data structure lacked the ability to properly secure tenant data from other tenants, it would be unlikely that any tenant would use the system. Consequently, a multi-tenant system must have some means to secure items in the database on a record-by-record basis.

Custom attributes, on their own, in no way implement any features of a multi-tenant data structure. Custom attributes, as disclosed by Lin, are in no way organized so that attributes for one tenant can be kept separate from other tenants. For example, referring to Fig. 3 of Lin, any user with access to any of the custom attribute tables would be able to access *every*

record associated with *every* custom attribute in the table. Lin discloses no way to distinguish between records of one tenant from the records of another tenant.

According to one aspect of the Applicant's invention, even when tenants do have access to the same table (or even the same columns within a given table), they do not have access to other tenants' data. In one disclosed embodiment, this is accomplished using "Org id" and "acc id" fields. (*see, e.g.*, Figs. 3, 5, and 7) One tenant with access to a given table will only be able to access records that belong to that tenant. This will be the case even if data from other tenants reside in the same table or even the same column. Lin simply does not disclose or suggest this fundamental aspect of multi-tenant functionality.

Additionally, users and their associated permissions are not the same as tenants and their associated permissions. The Applicant's disclosure states "[w]hile each user's sales data might be separate from other users' sales data regardless of the employers of each user, some data might be organization-wide data shared or accessible by a plurality of users or all of the sales force for a given organization that is a tenant. Thus, there might be some data structures managed by MTS 16 that are allocated at the tenant level while other data structures might be managed at the user level." (¶0032) Tenants and users are two distinct classifications in a multi-tenant system each with their own distinct set of applicable rules and permissions. A given entity using a multi-tenant database will typically have access to their data because of their status of a tenant. For example, a user from company A will not have access to data from company B, and vice-versa. *Additionally*, an entity will typically have certain permissions within their own data based on their status as a user. For example, an administrator may have full read/write permissions on all data and tables while other users may have more limited privileges based on their data needs. Combining these two features, an administrator with full privileges for data from company A will typically have no privileges to even see data from company B despite that fact that the data from company B resides in the same system and may even reside in the same table and columns as company A.

Govind also does not teach anything relating to a multi-tenant data structure and the Examiner does not cite Govind as teaching this particular element of the claims. As such, Govind cannot address the deficiencies in the disclosure of Lin.

Lin and Govind do not consider, let alone disclose or teach, a multi-tenant data structure. It is therefore respectfully submitted that claims 1-26 are patentable over Lin and Govind for least the foregoing reasons. Withdrawal of the present rejections of claims 1-26 is respectfully requested.

New claims 27-29

New claims 27-29 have been added to the application. These claims are dependent from existing independent claims, and so claims 27-29 are allowable over the cited art for at least the same reasons as the independent claims. Support for these claims can be found at least at ¶36 and 41.

Claims 27-29 recite that the first data type and the second data type of the tenant defined data type are data types that are native to the data structure. These claims are further allowable over Lin in view of Govind because they help make clear that the data types are *defined* by a tenant rather than *created* by a tenant. References such as Govind deal with "data types *whose native structure is not known to the database system.*" [Emphasis Added] (Abstract) The disclosure of Govind reveals that Govind is dealing with complex, custom data fields that may be a combination of many basic data types. For example, ¶¶18-23 of Govind show a custom data field that is a combination of a number and a date. In order to handle such complex data fields, Govind requires the data to be pickled and unpickled as it is stored in the database.

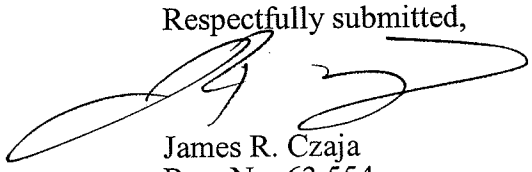
Claims 27-29 make is clear that the custom data fields do not need to be pickled or unpickled in order for the data structure to handle the tenant-defined data. The data structure is able to natively handle fields such as text, number, date, picklist, etc. As such, the data structure is able to manipulate this data without the use of any tenant-defined pickling routines to accomplish tasks such as indexing, sorting, etc. This difference helps to clarify a difference between created data fields (as disclosed by Govind) and defined data fields as is presently claimed in new claims 27-29.

CONCLUSION

In view of the foregoing, Applicants believe all claims now pending in this Application are in condition for allowance. The issuance of a formal Notice of Allowance at an early date is respectfully requested.

If the Examiner believes a telephone conference would expedite prosecution of this application, please telephone the undersigned at 415-576-0200.

Respectfully submitted,



James R. Czaja
Reg. No. 63,554

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 415-576-0200
Fax: 415-576-0300
J1C:j1c
62086607 v1